Job Scheduling for MapReduce

Matei Zaharia, Dhruba Borthakur*,
Joydeep Sen Sarma*, Scott Shenker, Ion Stoica

RAD Lab, *Facebook Inc
Motivation

• Hadoop was designed for large batch jobs
  – FIFO queue + locality optimization

• At Facebook, we saw a different workload:
  – Many users want to share a cluster
  – Many jobs are small (10-100 tasks)
    • Sampling, ad-hoc queries, periodic reports, etc

➤ How should we schedule tasks in a shared MapReduce cluster?
Benefits of Sharing

- Higher utilization due to statistical multiplexing
- Data consolidation (ability to query disjoint data sets together)
Why is it Interesting?

- Data locality is crucial for performance
- Conflict between locality and fairness
- 70% gain from simple algorithm
Outline

• Task scheduling in Hadoop
• Two problems
  – Head-of-line scheduling
  – Slot stickiness
• A solution (global scheduling)
• Lots more problems (future work)
Task Scheduling in Hadoop

- Slaves send heartbeats periodically
- Master responds with task if a slot is free, picking task with data closest to the node
Problem 1: Poor Locality for Small Jobs

## Job Sizes at Facebook

<table>
<thead>
<tr>
<th># of Maps</th>
<th>Percent of Jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 25</td>
<td>58%</td>
</tr>
<tr>
<td>25-100</td>
<td>18%</td>
</tr>
<tr>
<td>100-400</td>
<td>14%</td>
</tr>
<tr>
<td>400-1600</td>
<td>7%</td>
</tr>
<tr>
<td>1600-6400</td>
<td>3%</td>
</tr>
<tr>
<td>&gt; 6400</td>
<td>0.26%</td>
</tr>
</tbody>
</table>
Problem 1: Poor Locality for Small Jobs

Job Locality at Facebook

- Percent Local Maps
- Job Size (Number of Maps)

- Node Locality
- Rack Locality
Cause

- Only head-of-queue job is schedulable on each heartbeat
- Chance of heartbeat node having local data is low
- Jobs with blocks on X% of nodes get X% locality
Problem 2: Sticky Slots

• Suppose we do fair sharing as follows:
  – Divide task slots equally between jobs
  – When a slot becomes free, give it to the job that is farthest below its fair share
Problem 2: Sticky Slots
Problem 2: Sticky Slots

Problem: Jobs never leave their original slots
Calculations

Locality vs. Concurrent Jobs in 100-Node Cluster

- X-axis: Number of Concurrent Jobs
- Y-axis: Expected Node Locality (%)
Solution: Locality Wait

- Scan through job queue in order of priority
- Jobs must wait before they are allowed to run non-local tasks
  - If wait < $T_1$, only allow node-local tasks
  - If $T_1 < \text{wait} < T_2$, also allow rack-local
  - If wait > $T_2$, also allow off-rack
Locality Wait Example

Jobs can now shift between slots

<table>
<thead>
<tr>
<th>Job</th>
<th>Fair Share</th>
<th>Running Tasks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Job 1</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>Job 2</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
### Evaluation – Locality Gains

<table>
<thead>
<tr>
<th>Job Type</th>
<th>Default Scheduler</th>
<th>With Locality Wait</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Sort</td>
<td>2%</td>
<td>50%</td>
</tr>
<tr>
<td>Small Scan</td>
<td>2%</td>
<td>50%</td>
</tr>
<tr>
<td>Medium Scan</td>
<td>37%</td>
<td>98%</td>
</tr>
<tr>
<td>Large Scan</td>
<td>84%</td>
<td>99%</td>
</tr>
</tbody>
</table>
Throughput Gains

The graph shows the normalized running time for different tasks: Small Sort, Small Scan, Medium Scan, and Large Scan, comparing Default Scheduler and With Locality Wait. The gains are indicated as follows:

- Small Sort: 18%
- Small Scan: 20%
- Medium Scan: 70%
- Large Scan: 31%
Network Traffic Reduction

Network Traffic in Sort Workload

With locality wait
Without locality wait
• When is it worthwhile to wait, and how long?

• **For throughput:**
  – Always worth it, unless there’s a hotspot
  – If hotspot, prefer to run IO-bound tasks on the hotspot node and CPU-bound tasks remotely (rationale: maximize rate of local IO)
Further Analysis

• When is it worthwhile to wait, and how long?

• For response time:
  \[ E(\text{gain}) = (1 - e^{-w/t})(D - t) \]

  – Worth it if \( E(\text{wait}) < \text{cost of running non-locally} \)
  – Optimal wait time is infinity
Problem 3: Memory-Aware Scheduling

a) How much memory does each job need?
   – Asking users for per-job memory limits leads to overestimation
   – Use historic data about working set size?

b) High-memory jobs may starve
   – Reservation scheme + finish time estimation?
Problem 4: Reduce Scheduling

- Maps
  - Job 1
  - Job 2

- Reduces
  - Job 1
  - Job 2
Problem 4: Reduce Scheduling

Job 2 maps done
Problem 4: Reduce Scheduling

Maps

Time

Reduces

Time

Job 2 reduces done

Job 1
Job 2
Problem 4: Reduce Scheduling

- Job 1
- Job 2
- Job 3

Maps

Reduces

Job 3 submitted

Time
Problem 4: Reduce Scheduling

Maps

Time

Job 3 maps done

Reduces

Time

Job 1

Job 2

Job 3
Problem 4: Reduce Scheduling

Maps

Time

Reduces

Time

Job 1

Job 2

Job 3
Problem 4: Reduce Scheduling

Problem: Job 3 can’t launch reduces until Job 1 finishes
Conclusion

• Simple idea improves throughput by 70%
• Lots of future work:
  – Memory-aware scheduling
  – Reduce scheduling
  – Intermediate-data-aware scheduling
  – Using past history / learning job properties
  – Evaluation using richer benchmarks